长空御风
雷达站 Livox
开源报告

王书钰 1134643765@qq.com
刘建航 1358446393@qq.com
南京航空航天大学 长空御风

目录

1. 项目介绍

本项目为南京航空航天大学 RoboMaster2022 的雷达站项目，为了交流技术，推动 Livox 的应用和发展，促进雷达站兵种的提升，现针对雷达站的 Livox 部分进行一些详细说明。

我们使用 Livox Mid70 激光雷达，与双工业相机进行信息融合，对车辆进行识别与定位。同时得益于 Mid70 的超高精度，我们实现了哨兵辅助瞄准、敏感区域预警等附加功能，取得了不错的效果。

图2-1　雷达站功能定义

表 1：雷达站传感器类型

传感器	型号
左右相机	MV-SUA134GC-T　8mm　1:2.0　1/1.8”
上相机（不参与解算）	Realsense D435i
激光雷达	Livox Mid70

2. 具体实现

2.1 系统架构

系统层级:

整个雷达站采用 ROS（机器人操作系统）开发，代码遵循 ROS 架构。分为 4 层：硬件驱动层，数据处理层，功能逻辑层和前端显示层。

第三方中间件及模块:

CUDA_10.2, TensorRT, Tensorrtx, yolov5-6.0，OpenCV-4.5.4, Cmake

另一些驱动程序，以 ROS 功能包的形式运行：Livox_ROS_Driver, MindVision_ROS_Driver, RealSense_ROS_Driver,ROS_Serial

开发及运行环境:

Ubuntu20.04(操作系统), CLion(软件开发环境)

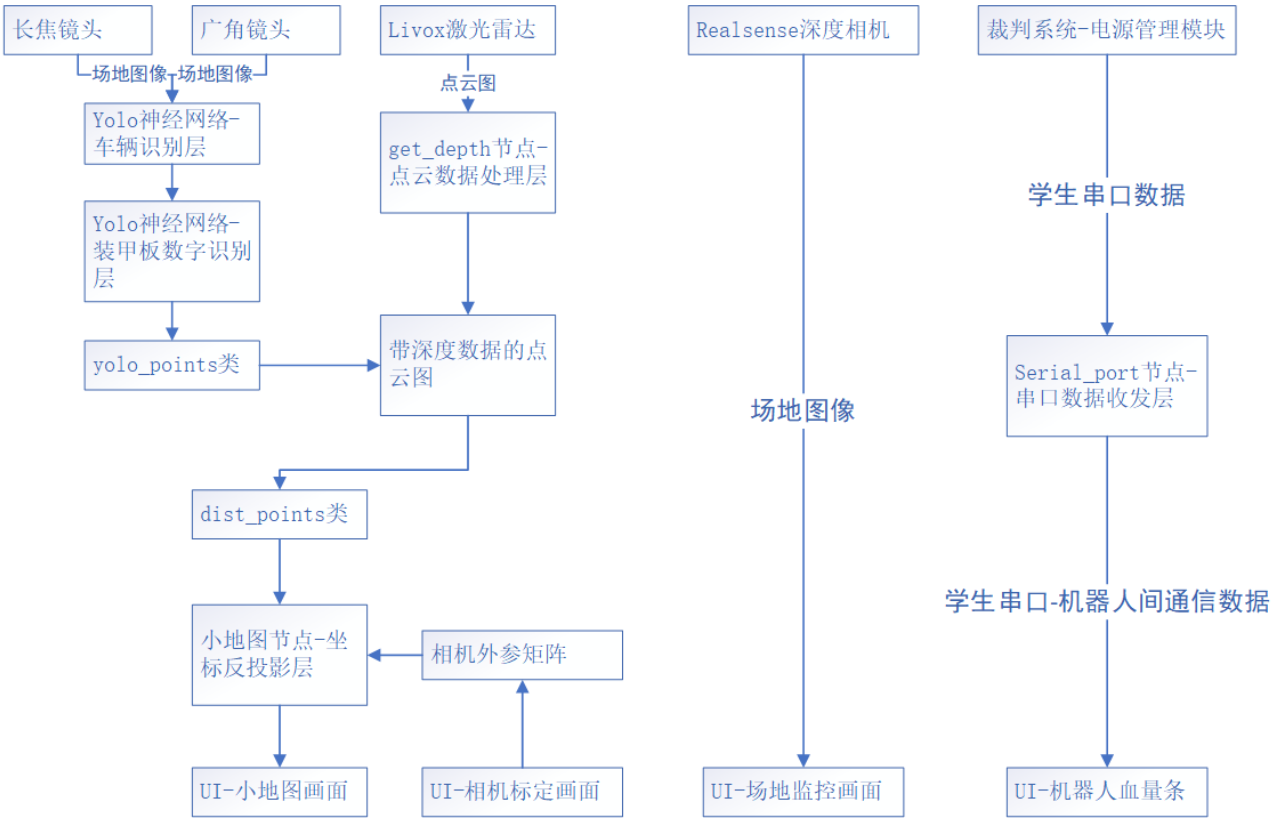

图2-2 运行流程图

2.2 小地图反投影

小地图算法的流程可以概括为：世界坐标系中一点的坐标向相机坐标系中投影（正投影过程）+投影点使用深度 d 和内外参矩阵的逆 inv[R,T], invM 向世界坐标系反投影的过程（反投影过程）。其中正投影过程已经在相机的成像过程中完成，而反投影过程则需我们编写算法，使其在运算平台端的计算中完成。这其中亦包括对深度 d、inv[R,T]、invM 的获取和计算，我们将这三个参数称为投影参数。因此，小地图算法的核心步骤即为两个：1.投影过程。2.投影参数的获取及计算。

下面，我分别介绍这两个核心步骤的原理及实现。

1.正投影过程和反投影过程有着数学上互逆的关系，只要理解了正投影过程的公式表示，就可以推出反投影过程。这里以正投影过程为例来讲解。正投影的过程可概括为：世界坐标系->相机坐标系->成像平面坐标系->像素坐标系。

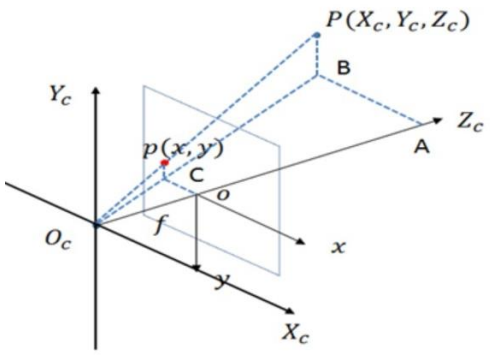

图2-3　小孔成像相机投影示意图

如图 2-1 所示为一个小孔成像相机模型。相机坐标系 $X_cY_cZ_c$，成像平面 uv，相机坐标系的 Zc 轴与相机光轴重合。有真实世界中一点 P，其在相机坐标系下的坐标为(Xc,Yc,Zc)，则根据小孔成像原理，P 点与相机坐标系原点 Fc 的连线与成像平面坐标系 xy 交于点 P'（坐标为(x,y)），此即为真实世界中的点 P 在相机成像平面上的二维投影点 P'。由几何学关系可得：

$$\frac{O_cO}{Z_c} = \frac{OC}{BA} => \frac{f}{Z_c} = \frac{x}{X_c} => xZ_c = fX_c \tag{1.1}$$

同理可得

$$yZ_c = fY_c \tag{1.2}$$

将其转化为矩阵关系式，即为

$$Z_c\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}f&0&0&0\\0&f&0&0\\0&0&1&0\end{bmatrix}\begin{bmatrix}X_c\\Y_c\\Z_c\\1\end{bmatrix} \tag{1.3}$$

之后，我们将 P’从成像平面坐标系 xy 转化至像素坐标系 uv，需对 x 轴和 y 轴分别乘以每毫米像素点数(dpm)，并加上成像平面中心点相对 u 轴和 v 轴的偏移量 u_0 和 v_0.即

$$p(u,v)=p\left(\frac{x}{dx}+u_0,\frac{y}{dy}+v_0\right) \tag{1.4}$$

其中 1/dx 和 1/dy 分别为 x 轴每毫米像素点数、y 轴每毫米像素点数。

将公式 1.4 转化为矩阵乘法形式并与 1.3 合并可得

$$\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}f_x&0&u_0&0\\0&f_y&v_0&0\\0&0&1&0\end{bmatrix}\begin{bmatrix}X_c\\Y_c\\Z_c\\1\end{bmatrix} \tag{1.5}$$

其中 fx=f/dx, fy=f/dy. 至此我们完成了相机坐标系->成像平面坐标系->像素坐标系之间的转换。为方便将公式 1.5 中的系数矩阵记为相机内参矩阵 M。

现在，我们假设世界坐标系为原点在图 2-2 的左下角，向右为 x 轴，向上为 y 轴，向屏幕外为 z 轴。此世界坐标系记作 $X_wY_wZ_w$.

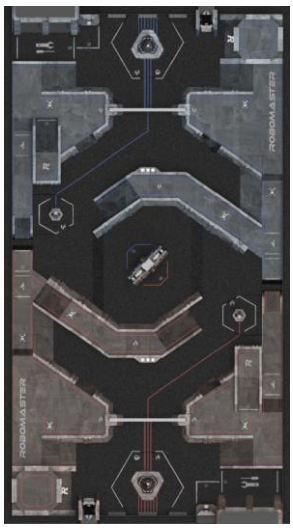

图2-4 赛场俯视图

根据空间直角坐标系之间的线性变换关系，$X_wY_wZ_w$ 到 $X_cY_cZ_c$ 的变换关系为

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = R\begin{pmatrix} x_{\text{world}} \\ y_{\text{world}} \\ z_{\text{world}} \end{pmatrix} + T = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}\begin{pmatrix} x_{\text{world}} \\ y_{\text{world}} \\ z_{\text{world}} \end{pmatrix} + \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}$$
$$= \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix}\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{1.6}$$

其中，R 记作两个坐标系之间的旋转矩阵，t 记作平移矩阵。

综合公式 1.5 和 1.6，我们可以写出世界坐标系到像素坐标系之间的变换关系

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M \cdot \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \cdot \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{1.7}$$

公式 1.7 即为世界坐标系到像素坐标系之间的转换关系，亦即正投影过程。

由线性代数理论推导，可得由世界坐标到像素坐标之间的转换关系（即反投影过程）为

$$\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = invR \cdot \left(invM \cdot Z_c \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - t \right) \tag{1.8}$$

其中 invR 是 R 的逆矩阵，invM 是 M 的逆矩阵，Zc 是点 P 在相机坐标系 Zc 轴的截距，其近似等于通过激光雷达所求得的点 P 距离雷达光源点的深度 d。在实际的程序代码中，我们使用深度 d 代替 Zc，解算精度达到要求.

$$\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = invR \cdot \left(invM \cdot d \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - t \right) \tag{1.9}$$

2.投影参数的获取与计算

从公式 1.9 中可见须获得的参数是 invR、t、invM、d。下面分别介绍他们的获取方法。

2.3 相机标定

我们使用了 ROS 下的 camera_calibration 包标定相机内参。相比于 OpenCV 或者 Matlab 中的标注方式，camera_calibration 的显著优点是自动化程度极高。通过进度条指引标图者调整标定板的角度，并且无需手动拍照，短时间内便可以自动拍摄和解算大量的图片。操作的方便间接的提高了标定的精度。通过这种方式便可以获得相机较为准确的内参，

$$M = \begin{matrix} f_x & o & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{matrix}$$

进而获得其逆矩阵invM。

2.4 位姿估计

旋转和平移矩阵 R、t 由我们在赛前 3 分钟内使用 solvePnP 方法临场标定获得。我们使用经典的四点法，事先选取好赛场上的 4 个标定点，根据规则手册计算出以式 1.9 中 $X_wY_wZ_w$ 为基准的世界坐标，在赛前 3 分钟内通过 ui 选点操作获取其像素坐标，输入 OpenCV 库的 cv::solvePnPRansac 函数并计算出旋转和平移矩阵 R、t。通过 cv::Rodrigues 将 R、t 转化为罗德里格斯形式，并求解其逆矩阵 invR。

2.5 多传感器融合

多传感器融合的过程简单来说就是使用事先标定好的雷达与相机之间的旋转平移矩阵将车辆识别框投影至点云图中，对每个投影框内部的点的深度求取距离统计值（依照点个数多少分别取不同的统计值），得到该车辆相对雷达站相机的距离 d，将距离 d 和车辆在相机画面中的二维坐标打包发送至小地图节点。具体实现如下。

联合标定

首先，我们使用了 livox 官方提供的激光雷达标定方式（https://github.com/Livox-SDK/livox_camera_lidar_calibration）进行手动标定。在拍摄了多角度的图片并录制点云 rosbag 后，分别使用鼠标在点云图和彩色图中按照相同的顺序点出四个角点，通过 PNP 解算即可获得旋转和平移矩阵。

由于为手动标注，工作的繁琐注定了图片和点云图的数量不会太多，同时由于手动选点的误差，使得得到的参数注定不会十分精确。所以还需要想办法进一步提高联合标定的精度。

我们使用了香港大学开源的无目标场景下激光雷达和相机自动标定（https://github.com/hku-mars/livox_camera_calib）的方法。该方法可以自动提取深度图和彩色图中共同特征，并通过迭代的方式，使得误差达到最小。通过这种方式，我们进一步提高了联合标定的精度。

最终我们获得了从雷达坐标系到像素坐标系的旋转和平移矩阵 R、t。

点云更新策略

在 Livox 官方提供的 ROS 驱动中，有设置积分时间的参数，但积分时间过长，势必导致消息发送频率下降，进而引起实时性降低。为了能够有足够多的点云，并且能够实时获取最新的点云数据，我们在程序中设置了点云队列：保留在过去一段时间内的全部点云，当点云数量达到最大值时，抛弃旧点云。

点云投影

为了获取深度图并将深度图与彩色图对齐，需要对点云进行反投影操作：

$$\begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \cdot \begin{pmatrix} X_L \\ Y_L \\ Z_L \\ 1 \end{pmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \cdot Z_c \tag{2.1}$$

式中 X_L、Y_L、Z_L 为雷达坐标系下的坐标。

通过这样的运算，我们即可得到与彩色图对齐的深度图。

每一次收到点云，都会获得一张深度图。这样，我们便从点云队列中获取到了深度图队列。

深度获取

从最新的深度图开始寻找，遍历在 ROI 中的点。如果结果为空，则在队列中更久的深度图中寻找，直到结果不为空。使用结果深度的统计值即可获得目标深度。如果在队列中没有位于 ROI 中的点，则返回空值。

通过上面这些操作，我们实现了传感器的融合，获得了从像素坐标求解世界坐标最为关键的参数：d。

2.6 哨兵辅助瞄准

在实战中我们发现，当有敌方车辆进入哨兵的识别范围时，哨兵往往由于巡航策略问题，很长时间才能发现敌方，错过了最优的打击时机。最直接的改进方案当然是寻找最优巡航策略，但是我们本赛季另辟蹊径，使用雷达指引哨兵快速定位目标。

通过将距离哨兵最近的敌方车辆坐标与哨兵坐标相减，即可获得地方车辆与哨兵的相对坐标。通过车间通信，即可实现哨兵辅助瞄准。

最终实现的效果令人满意。在禁用哨兵自身的视觉自瞄条件下，完全使用雷达站的视觉识别方案可将哨兵的自瞄精度做到 1m 以内（哨兵红点距车辆的距

离误差），在某些特定区域（如 R0 高地等距离我方较近的位置），甚至可以实现仅依靠雷达锁定敌人装甲板。这个精度已经可以令哨兵根据雷达站提供的方位将云台快速瞄向目标的大致位置，再使用自身的自瞄系统对目标进行精确的打击。大大缩短了哨兵巡航的不确定性。由于时间限制，并未测试辅助自瞄与自身自瞄共同工作时的工况，可以预计到会出现哨兵只使用辅助自瞄而弃用自身自瞄的误决策问题，这类问题有待进一步的测试与优化。

2.7 英雄辅助测距

英雄与前哨站距离测量误差在 1.5 米以内。对于英雄的吊射重力补偿程序来讲，这个误差还不够小。我们尝试在距离解算结果上加上一个固定补偿量来修正，结果是对于同一场比赛，修正后的测算距离基本能够保持在 1m 以内的误差，这个结果已被控制在重力补偿程序允许的输入误差范围内，因此我们的优化方案就是在每场比赛开始前，快速地测出固定补偿量的值并传给英雄控制程序，使其在本场比赛中可以使用精度较高的距离值来吊射。

3. 代码仓库

https://github.com/nuaa-rm/radar_station2022

觉得有用的话记得点个 star 😊。

4. 参考文献

1.高翔、张涛：《视觉 SLAM 十四讲》

2. 坐标系变换 & 坐标变换 - 赖东风的文章 - 知乎 https://zhuanlan.zhihu.com/p/274976956

3.OpenCV-4.5.4 文档

4.Robomaster 2022 超级对抗赛比赛规则手册

5.裁判系统学生串口手册 V1.3

6.tensorrtx-GitHub 仓库及其使用方法：https://github.com/wang-xinyu/tensorrtx

7.上海交通大学 RM2021 雷达站开源报告

长空御风 雷达

王书钰 1134643765@qq.com
刘建航 1358446393@qq.com
南京航空航天大学 长空御风